Efficient Organization of Digital Periphery to Support Integer Datatype for Memristor-based CIM

Mahdi Zahedi, Mahta Mayahinia, Muath Abu Lebdeh, Stephan Wong, Said Hamdioui
Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands
Email: {m.z.zahedi, m.mayahinia-1, m.f.m.abuLebdeh, j.s.s.m.wong, s.hamdioui}@tudelft.nl

***Abstract*—Von Neumann-based architectures suffer from costly communication between CPU and memory. This communication imposes several orders of magnitude more power and performance overheads compared to the arithmetic operations performed by the processor. This overhead becomes critical for applications that require processing a large amount of data. Computation-in-Memory (CIM) leveraging memristor devices in the crossbar structure offers a potential solution to tackle this challenge. However, support for the integer data type is lacking in CIM approaches as most solutions operate on a single/few bits only. This paper proposes a new organization of the periphery (next to memristor crossbar) to compute matrix-matrix multiplication (MMM) at the tile level. More precisely, the analog additions performed in the crossbar is complemented with additions performed in the digital periphery. In this mixed analog-digital system, digital additions are performed in a way that only the minimum size of adders are required - this is to reduce the latency of the digital periphery as much as possible. In addition, the design is customized to the number of ADCs as well as datatype sizes to support different possible scenarios. The results show that our organization reduces energy and latency up to $50\times$ and $3\times$, respectively, compared to the reference design.**

I. INTRODUCTION

Big data applications such as neural networks and databases are being widely used in different domains. These applications require processing of large amounts of data that is usually located far away from the processing units. Consequently, the transport of data between the processing unit(s) and the memory leads to huge energy consumption and long latencies. This greatly diminishes the usefulness of von Neumann computers (to process the data) [1]–[3]. A solution to overcome this is to employ the Computation-in-Memory (CIM) approach that proposes the utilization of (new) technologies that allows for both storage and computing within the same (storage) structure. Such technologies include: resistive RAM (ReRAM) [4], phase-change memory (PCM) [5], and spin-transfer torque magnetic RAM (STT-RAM) [6]. Furthermore, they provide additional benefits such as high density, near-zero standby power, and non-volatility [7]. Although the memory crossbars built from these devices are usually low power, the peripheral circuits needed to control the crossbar can completely alleviate the gains. As an example, many applications such as Neural Networks (NNs) and Convolutional Neural Networks (CNN) at least require to integer (number) computations. However, since only limited levels can be stored in one memristor cell, the (bit-)vector representing the number should be distributed over multiple cells, which requires some extra processing outside the crossbar to get a meaningful result. Therefore, great care must be taken in the design of these peripheral circuits to really deliver efficient in-memory computing systems.

Recent researches in the field of CIM can be categorized as follows. First, the majority of researchers are focusing on improving device characteristics such as performance, energy, and endurance considering different technologies [8], [9]. Second, some works investigate different ways to perform computation inside the crossbar [10]–[12]. Third, researchers are focusing specifically on mapping NN and CNN applications to memristor crossbars to serve as accelerators for these applications. [13]–[15]. However, no implementation details are provided how to deal with the post-processing of integer datatypes in the digital peripheral circuits surrounding the crossbar. Until now and to the best of our knowledge, only a single publication has addressed this challenge [16]. In this work, the authors perform the first stage of matrix-matrix multiplication in the crossbar and continue performing the remaining additions in the analog domain through use of analog buffering and accumulation units. The advantage is the reduction of analog-to-digital (ACD) operations, but at the cost of a higher-precision ADC unit. This is worsened when considering larger crossbars or larger integer word sizes. Moreover, analog computing is more fragile to noise compared to digital computing, which is critical for applications that require high accuracy.

In this paper, in order to support the unsigned integer datatype, we propose a new adder structure in the periphery next to the memory crossbar. The proposed design utilizes minimum-sized adders customized based on technology-driven restrictions, e.g., the number of active crossbar rows per addition. Furthermore, our approach takes into account the number of ADCs employed by the crossbar. Finally, the structure is not restricted to a single fixed integer word size and can flexibly support different word sizes without hardware changes. The contributions of this paper are the following:

- a new design is proposed to aid an in-memory crossbar to perform additions targeting integer matrix-matrix multiplication (MMM).
- the proposed design is customizable in order to support a varying number of ADCs.
- the proposed design is evaluated in terms of energy and performance using our in-house simulator configured with the parameters defined by technology and synthesized designs.

2159-3477/20/$31.00 ©2020 IEEE
DOI 10.1109/ISVLSI49217.2020.00047

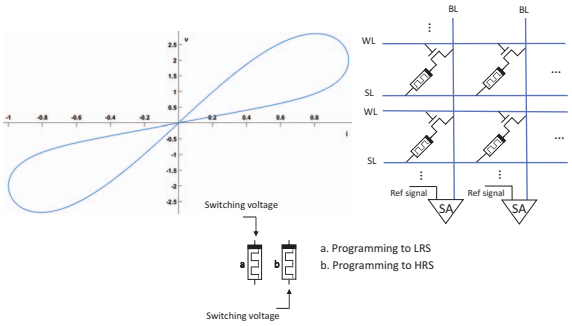

Fig. 1: Pinched hysteresis v-i curve considered as Memristor's fingerprint [17], 1T-1R crossbar structure and Bipolar switching

The remainder of this paper is organized as follows. In Section II, the state-of-the-art which is categorized in different levels of CIM abstraction will be explained briefly. In addition, the way that analog computation performed in the memristor crossbar is described briefly. Section III presents the organization of the addition unit. In Section IV, the evaluation model and the experimental results will be discussed. Finally, Section V concludes the paper.

II. BACKGROUND

A. Memristor devices

Memristors are devices that could achieve multiple distinct resistive levels; in the case of 2 resistive levels – High or Low – either of these sates can represent one of the binary values '0' or '1'. Once the device obtains either of the states, it would be able to keep its state for a relatively long period of time without consuming energy, therefore, memristors are considered as non-volatile memories (NVMs). Switching a device state from one state to the other can be performed in either a unipolar or a bipolar manner; in unipolar switching, the specific values of voltage (or current) are used to make the device to switch and in the bipolar switching, different polarity of voltage (or current) needs to be applied to the device in order to switch. To read the data from memristive devices, a signal (voltage or current) is applied to the cell and the value of the output signal determines the data stored in a cell. The read signal must be small to prevent the cell from being disturbed. Memristors can be used in a crossbar network with a structure of 1T1R. Fig 1 shows the I-V curve of the memristor which is a pinched hysteresis under sinusoid input [17], a crossbar structure, and bipolar switching.

B. Memristor-based computation circuits

Computation-in-Memory (CIM) is an approach to overcome the memory wall problem by reducing the distance between data and computation within the same memory array by exploiting new memory technologies. Within this approach, we can clearly distinguish two directions by looking at where the results of the computation is being produced. CIM-A is used to denote when the results are produced within the memory array and CIM-P is used to denote when the results are produced in the periphery [18]. A brief description of CIM-A and CIM-P are presented in the following:

- *CIM-A*: The main advantage of CIM-A is the achievable high bandwidth of computations as data is stored and computed within the same memory array. However, storing the results in the same memory also means an increased amount of write operations within the array. The latter is detrimental due to the limited endurance of current-day NVMs. Moreover, performing computations within the array requires significant modifications to the (analog) array as well as the (digital) periphery. [19], [20] are examples of CIM-A approaches.
- *CIM-P*: In comparison to CIM-A, the computational bandwidth of CIM-P approaches is reduced – still, it is much higher than approaches that need to transfer the data out of the memory tile. However, as less writes to the memory array are needed, the endurance issue can be avoided. Furthermore, the memory array requires far fewer modifications. Moreover, changes in the (digital) periphery are easier to implement.

C. In-memory architecture

A few promising studies design application-specific accelerators enjoying CIM concept mostly for NNs. ISAAC [13] designs an accelerator for convolutional neural networks using ReRAM devices to exploit the efficiency of in-memory computing. PRIME [14] targets NN as an application and proposes an architecture that enables the authors to configure some portion of memory as a computation unit according to the requirements. To control the memristor-based memory, new in-memory instructions were proposed [15] based on the SIMD execution model. However, none of the aforementioned works provide implementation details to show how to support integer MMM in the tile-level.

To the best of our knowledge, CASCADE [16] is the only work providing implementation details to support integer numbers for MMM. In this work, by employing analog buffers, some parts of partial sum and accumulation happen in an analog way, which helps to reduce the number of ADC conversions. Although the number of conversions are decreased leveraging this approach, the resolution of ADCs should be increased, which imposes energy overhead that grows exponentially [13]. Furthermore, as the number of columns contribute to the analog partial sum and accumulation increased, the relative difference between voltage/current levels of output reduced, which requires more complex ADCs. Equations 1 and 2 demonstrate a set of equivalent resistance for one column as well as several columns when they are attached to each other, respectively.

$$R_{eq} \in L = \{R_{off}, R_{on}, R_{on}/2, ..., R_{on}/n\} \quad n : number\,of\,crossbar\,rows \tag{1}$$

$$R_{eq} \in L = \{R_{off}, R_{on}, R_{on}/2, ..., R_{on}/(m*n)\} \quad n : number\,of\,crossbar\,rows \quad m : number\,of\,connected\,columns \tag{2}$$

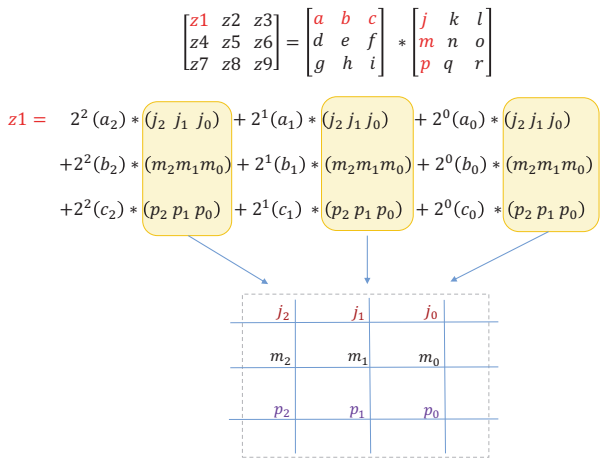

Fig. 2: Mapping of integer number into the crossbar

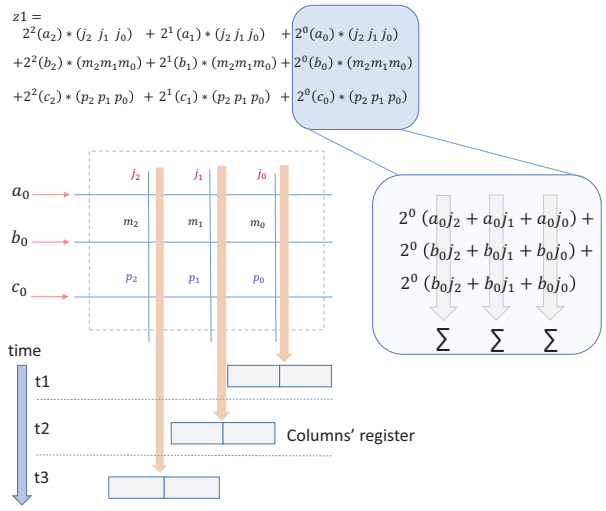

Fig. 3: Analog addition inside the crossbar

According to the equations, the relative difference between resistance levels is reduced more as the datatype size increases. In addition, despite the reference architectures that can perform precise MMM, it was assumed that approximation can be tolerated to be able to remove ADCs for some columns. Finally, besides the complexity for placement and routing of many analog buffers considered in the periphery, the maximum current that can be tolerated according to the technology constraint has to be taken into account.

III. Overall Addition Unit Organization

In this section, the proposed periphery design to support integer MMM is explained. First, the mapping of data to the crossbar is discussed. Subsequently, the digital processing which has to be performed on the output of the crossbar, to prepare the final result of MMM, is presented. This processing has to be performed in several steps, which depends on the number and precision of ADCs used in the CIM tile. We elaborate on different scenarios to see what changes are required for each.

A. Integer data mapping

In the following, we will utilize a simplified MMM example in order to highlight the mapping of the multiplicands in the crossbar. We assume that the calculation of a single element $z1$ (in the result matrix) is as follows: $a*j+b*m+c*p$. All values are assumed to be 3 bits in this example. The values a, b, and c are the multipliers and j, m, and p are the multiplicands. Furthermore, we assume that the multiplicands are needed again in other MMM operations and this is the reason to map them into the crossbar - this avoids multiple writes to the crossbar. Figure 2 depicts the multiplication (for $z1$) written out in full and the mapping of the multiplicands to the crossbar assuming that each cell can only store a single bit. It should be clear from the figure that we first multiply the zero-th bit of the multipliers with the multiplicands and sum them up as they pertain to the same weight (2^0) in the final result. The same multiplicands (indicated in yellow) are needed again when multiplying them with higher-order multiplier bits. In case the crossbar has more columns, the multiplicands k and l can also be mapped. Otherwise, different tiles are needed to map those multiplicands.

B. Proposed organization for addition units

Our proposed adder design clearly distinguishes three adder stages to perform the MMM operation. The design considerations of each stage are described in the following.

Stage 1: Adding one column

In this stage, the addition within a single column is performed - indicated by the orange-brown arrow(s) in Figure 3. Essentially, the addition in this stage is performed in an analog manner - using Kirchhoff's Law - if and only if (1) the analog-to-digital converter (ADC) is capable of distinguishing between all the possible current levels and (2) all the necessary rows (related to the multiplier) can be activated at the same time - this is dictated by the utilized technology. If either (or both) of the mentioned conditions is (are) not met, the addition needs to be broken down to only adding those number of rows that can be supported by the ADC accuracy or technology. This means that the analog addition is only performed among a smaller number of rows and the resulting intermediate results need to be summed up together in the digital domain. The latter is depicted in Figure 4(b). It should be noted that the size of the digital adder here is determined by the largest possible results after all rows have been added - worst case: $log_2(rows)$. The result of this stage is the summation of all the terms related to one bit-position of the multiplier and one bit-position of the multiplicand.

Stage 2: Adding multiple columns

As the result of the first stage relates to the summation of only a single bit-position of the multiplicand, the second stage adds up all the terms (i.e., all bit positions) related to the multiplicand (multiplied with only one bit-position of the multiplier) - see blue box in Figure 3. In this stage, we are assuming that the multiple columns are sharing one ADC and that (for now) the integer word size (of multiplicands)

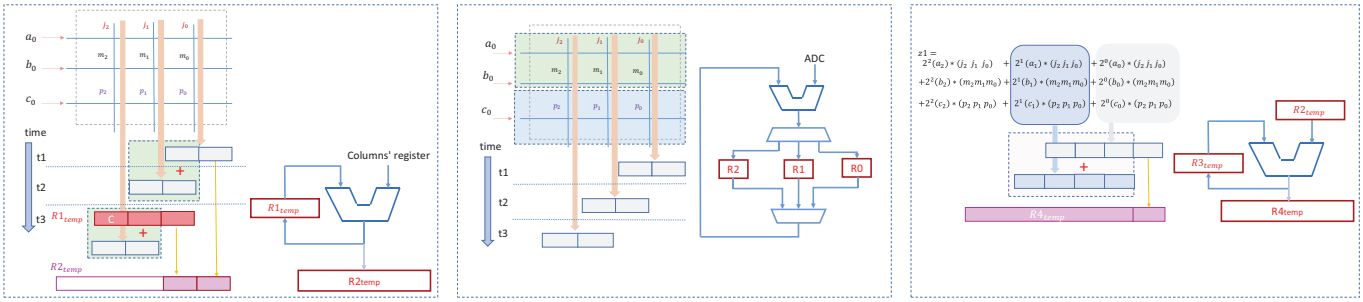

Fig. 4: Required digital processing phases per ADC

does not exceed this number. The first assumption is not a restriction as all operations described in this stage remain the same if assumption would not hold. The second assumption is a restriction as loosening it would require additional logic to combine the (intermediate) results. Since this would severely complicate the introduction of our approach at this stage, we defer it to a later section in this paper. The results of each column (see Figure 4(a)) relate to different weights of the multiplicand. Consequently, we can use the same adder (maximum size: $log_2(rows)$) to perform addition as long as each time the addition is performed, the intermediate result is shifted by one position. This means that the lowest significant bit, which is shifted out, can be stored in a temporary register ($R2_{temp}$). The higher order bits are stored in ($R1_{temp}$) - this register must be initialized to zero before the MMM operation is started. The aforementioned addition can be performed while the ADC is "scanning" the columns. The result of the second stage is a partial result of the MMM operation that relates to a single bit-position of the multiplier, e.g., a_0, b_0, and c_0.

Stage 3: Adding higher-order results

In this stage, the partial sums related to different bit positions of the multiplier need to be summed up. This is depicted in Figure 4(c). The partial sums related to 0-th bit position of the multiplier (gray box) need to added to the partial sum of the 1-st bit position of the multiplier (blue box), and so on. Here, we can employ the same adder structure as described in stage 2 - see Figure 4(a). Only now, the adder and needed temporary registers are larger in size.

Figure 5 puts together all the hardware discussed before for each processing phase. Following is the generalized size of the registers used in the proposed organization. Clearly, the size of the adders is equal to their corresponding registers.

$$R0 = R1 = R2 = R1_{temp} = \log_2(crossbar\ height) + \log_2(cell\ levels) \quad (3)$$

$$R2_{temp} = R3_{temp} = int\ size(multiplicand) + \log_2(crossbar\ height) \quad (4)$$

$$R4_{temp} = int\ size(multiplier) + int\ size(multiplicand) + \log_2(crossbar\ height) \quad (5)$$

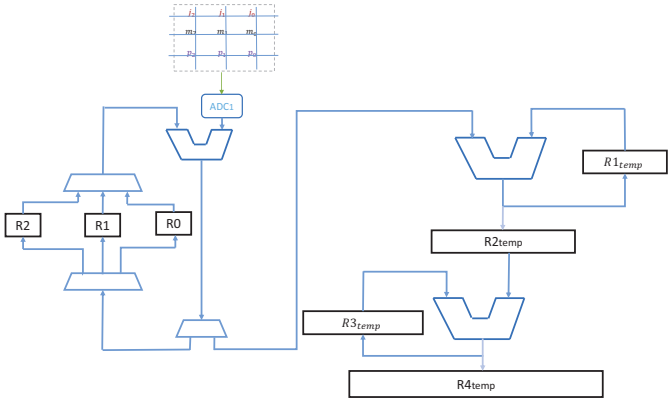

Fig. 5: The overall organization required per ADC

What we discussed so far, was based on the assumption that the size of integer numbers stored in the crossbar matches to the number of columns shared an ADC. However, two other possible scenarios have to be considered. First, by increasing the number of ADCs, a number stored in the crossbar might be split and distributed over multiple ADCs. To claritfy this further, Figure 6 illustrates this scenario in which 32-bit numbers distributed over 4 ADCs. Accordingly, to obtain the final result, the values stored in all the $R4_{temp}$ registers, employed for each ADC, have to be summed up. Although more hardware is required as the number of ADCs increased, the length of the registers and adders employed for processes per ADC are decreased.

IV. Evaluation and Discussion

In this section, we will evaluate our design against a reference design. First, we introduce the reference design and the target technology. Subsequently, the results regarding the performance and energy of our proposed design will be discussed.

A. Experimental setup

Reference architecture. In the reference design, we assume that a single adder to perform accumulation between shared columns and different bit positions of the multiplier are connected to each ADC. The size of the adder is fixed and must be chosen based on the largest possible value resulting from the MMM - it is specified in Equation 6. This means that the value produced by the ADC is merely an intermediate result

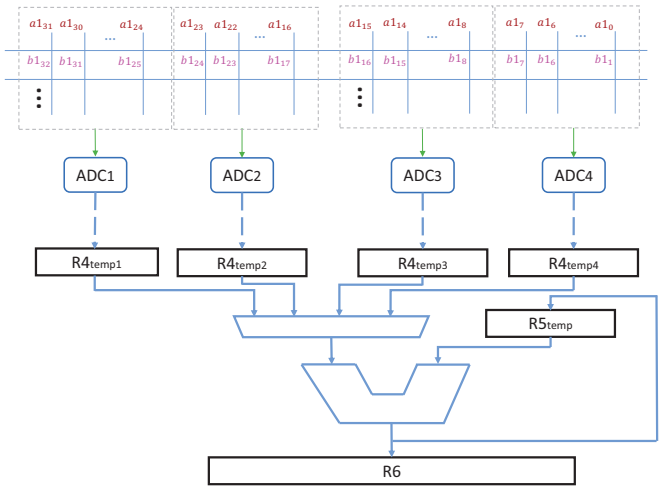

Fig. 6: The possible organization required between ADCs

TABLE I: Tile configuration

Crossbar		
Technology	ReRAM (256x256 @1bit)	
Read energy	0.4pj per cell	
Write energy	40pj per cell	
Read/Write latency	100ns	
ADC (8-bit)		
Energy	2pj per sample	
Latency	1ns per sample	
Carry-lookahead Adder		
Size	Energy (per computation)	Latency
8-bit	0.01pj	1ns
16-bit	0.03pj	2.2ns
24-bit	0.08pj	3.2ns
40-bit	0.25pj	5.6ns
72-bit	0.78pj	9.8ns

that must be summed up into the accumulator - remember that only a single bit of the multiplier is multiplied with the multiplicand and each ADC read-out corresponds only to a single bit-position of the multiplicand. Due to the previously stated manner of summation, the intermediate results must be shifted by the correct number of positions (based on the bit-positions of the multiplier and the multiplicand) before entering the adder.

$$\begin{aligned} &Adder\ size = \\ &\quad int\,size(multiplier) + int\,size(multiplicand) \\ &\quad + \log_2(crossbar\ height) \end{aligned} \tag{6}$$

In order to evaluate our design (against the reference design), we have synthesized both designs using Cadence Genus with standard cell 90nm UMC library to generate latency and energy numbers. The latency and energy numbers related to the crossbar (assumed to use ReRAM for now) and (SAR) ADCs are taken from [21] and [22], respectively. These numbers are summarized in Table I and more detail information about ReRAM device can be found in [21]. Subsequently, these numbers serve as input to our in-house tile simulator. Our simulator is capable of executing programs and capturing the behavior of the crossbar and peripheral circuit.

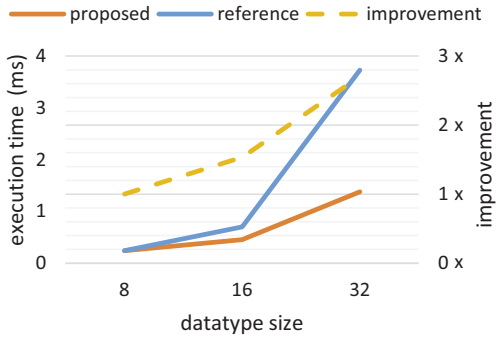

Fig. 7: Execution time for different integer datatype sizes

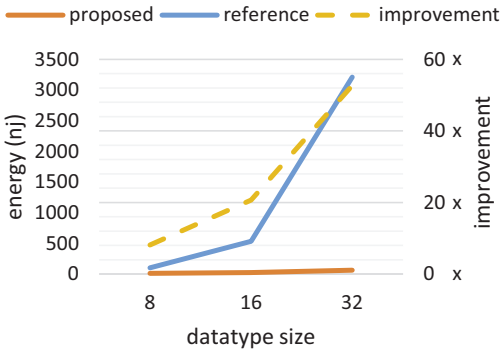

Fig. 8: Energy consumption of addition unit for different datatype sizes

As a benchmark, we have chosen the linear-algebra kernel "gemm" from the Polybench/C benchmark suite. In this kernel, first the multiplicands are written into the crossbar and then the actual multiplication is performed. We utilized our in-house developed compiler to translate the gemm benchmark into a sequence of in-memory instructions controlling the crossbar and its peripheral circuits.

B. Experimental results

Reading data from the crossbar's columns (read out phase) is inherently slow mainly because of the shared ADC between multiple columns and can be considered as the bottleneck of the architecture. However, as the size of adder grows, its latency can be dominant over the latency imposed by ADC. Considering the proposed design in our experiment, since the crossbar has 256 rows, the maximum size of the first two adders (see Figure 5) is always 8-bit regardless of datatype size. Therefore, there is no extra overhead on the latency of read out phase. Rather, in the reference design and according to the Equation 6, the required size of adder is much larger. Accordingly, considering a 1 ns latency for the ADC, as the size of adder increased more than 8-bit, it becomes the bottleneck of the system and makes the read out phase more costly.

Figure 7 depicts the execution time of the kernel (including writing to the crossbar). In this experiment, we assume that the size of the integer numbers matches the number of columns shared by one ADC. According to the figure, the size of the

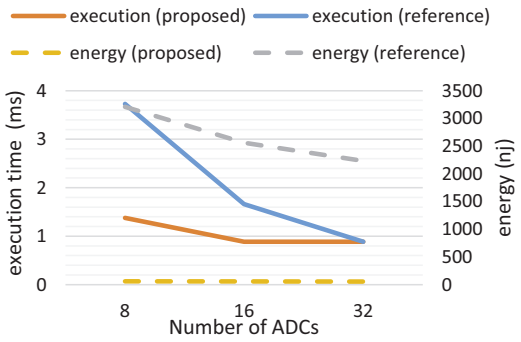

Fig. 9: Energy consumption of addition unit and execution time of the benchmark for different number of ADCs

first two adders for the proposed design are always constant (here 8-bit). Rather, as the datatype size increases, a larger adder has to be employed for the reference design. Considering an 8-bit datatype size in Figure 7, a 24-bit adder has to be used for the reference design, which imposes a 3 times bigger latency than an ADC. However, due to the abundance of ADCs, the entire readout phase is not the bottleneck of the system (see Table I for the crossbar latency). Therefore, there is no performance improvement at this point. Figure 8 shows the energy improvement achieved by the proposed design. Although the number of computations is always the same, they are performed with smaller adders, which has a positive impact on the energy consumption of the addition unit. Finally, the impact of the number of ADCs on the execution time and energy of the addition unit using 32-bit datatype size are presented in Figure 9. As the number of ADCs increased, the performance is improved. In addition, although more adders are employed (see Figure 6), their size is decreased, which leads to less energy consumption.

V. Conclusion

In this paper, we proposed a new organization for memristor-based crossbar periphery to support integer matrix-matrix multiplication at the tile level. Besides to the analog additions in the crossbar, digital additions are performed in a way that minimum adder sizes are required. The results show that using the proposed design, performance improved up to 3x and energy reduced by 50x compared to the reference design.

References

[1] S. Srikanth, L. Subramanian, S. Subramoney, T. M. Conte, and H. Wang, "Tackling memory access latency through dram row management," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '18. New York, NY, USA: ACM, 2018, pp. 137–147.

[2] "Calculating memory power for DDR4 SDRAM," https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf, 2017.

[3] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, Mar 2017.

[4] B. Govoreanu, G. S. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. J. Wouters, J. A. Kittl, and M. Jurczak, "10*10nm2 hf/hfox crossbar resistive ram with excellent performance, reliability and low-energy operation," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 31.6.1–31.6.4.

[5] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase change memory architecture and the quest for scalability," *Commun. ACM*, vol. 53, no. 7, pp. 99–106, Jul. 2010.

[6] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram," in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, Dec 2005, pp. 459–462.

[7] S. Yu and P. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, Spring 2016.

[8] G. W. Burr, M. J. Brightsky, A. Sebastian, H. Cheng, J. Wu, S. Kim, N. E. Sosa, N. Papandreou, H. Lung, H. Pozidis, E. Eleftheriou, and C. H. Lam, "Recent progress in phase-change memory technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 146–162, June 2016.

[9] S. Rashidi, M. Jalili, and H. Sarbazi-Azad, "A survey on pcm lifetime enhancement schemes," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 76:1–76:38, Aug. 2019. [Online]. Available: http://doi.acm.org/10.1145/3332257

[10] L. Xie, H. A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 176–181.

[11] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, Nov 2014.

[12] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 757–763.

[13] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[14] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[15] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 1–14.

[16] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "Cascade: Connecting rrams to extend analog dataflow in an end-to-end in-memory processing paradigm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 114–125.

[17] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[18] M. A. Lebdeh, U. Reinsalu†, H. A. D. Nguyen, S. Wong, and S. Hamdioui, "Memristive device based circuits for computation-in-memory architectures," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[19] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive'switches enable 'stateful'logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[20] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, no. 6, pp. 1165–1172, 2005.

[21] K. Fleck, U. Böttger, R. Waser, N. Aslam, S. Hoffmann-Eifert, and S. Menzel, "Energy dissipation during pulsed switching of strontium-titanate based resistive switching memory devices," in *2016 46th European Solid-State Device Research Conference (ESSDERC)*, Sep. 2016, pp. 160–163.

[22] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mw 8b 1.2 gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32 nm digital soi cmos," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, 2013.